Experiences from a Decade of Development

Philip Levis
Stanford University
Keynote, SESENA 2013
Back to 1999...

“Information technology (IT) is on the verge of another revolution... The use of EmNets [embedded networks] throughout society could well dwarf previous milestones.”

“The motes [EmNet nodes] preview a future pervaded by networks of wireless battery-powered sensors that monitor our environment, our machines, and even us.”

• **Idea: operating system for “sensor networks”**
  ‣ Microcontrollers (bah, virtual memory and 32-bit words)
  ‣ Low-power (2μA - 4mA)
  ‣ Wireless communication (good luck with that)
  ‣ Started as Perl scripts used by a handful of academics

• **13 years later...**
  ‣ ~25,000 downloads a year, hundreds of thousands of nodes!
  ‣ Worldwide community of hundreds of contributors!
  ‣ Hundreds of research papers!
  ‣ The Internet of Things!
This Talk

• Two design principles for embedded software
  ‣ Minimize resource use
  ‣ Structure interfaces and code to prevent bugs

• A technical result: static virtualization

• A lesson: avoid the island syndrome
TinyOS is the work of hundreds of contributors over a decade.
(of which I am only one, the core WG chair, who joined 18 months in)

This paper and talk are my personal opinions and observations.
This Talk

• Two design principles for embedded software
  ‣ Minimize resource use
  ‣ Structure interfaces and code to prevent bugs
• A technical result: static virtualization
• A lesson: avoid the island syndrome
Minimize Resource Use

<table>
<thead>
<tr>
<th>Model</th>
<th>ROM</th>
<th>RAM</th>
<th>Sleep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2002</td>
<td>1kB</td>
<td>128B</td>
<td>1.3μA</td>
<td>$0.94</td>
</tr>
<tr>
<td>F1232</td>
<td>8kB</td>
<td>256B</td>
<td>1.6μA</td>
<td>$2.73</td>
</tr>
<tr>
<td>F155</td>
<td>16kB</td>
<td>512B</td>
<td>2.0μA</td>
<td>$6.54</td>
</tr>
<tr>
<td>F168</td>
<td>48kB</td>
<td>2048B</td>
<td>2.0μA</td>
<td>$9.11</td>
</tr>
<tr>
<td>F1611</td>
<td>48kB</td>
<td>10240B</td>
<td>2.0μA</td>
<td>$12.86</td>
</tr>
</tbody>
</table>

TI MSP430 Microcontrollers
# Minimize Resource Use

<table>
<thead>
<tr>
<th>Model</th>
<th>ROM</th>
<th>RAM</th>
<th>Sleep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2002</td>
<td>1kB</td>
<td>128B</td>
<td>1.3µA</td>
<td>$0.94</td>
</tr>
<tr>
<td>F1232</td>
<td>8kB</td>
<td>256B</td>
<td>1.6µA</td>
<td>$2.73</td>
</tr>
<tr>
<td>F155</td>
<td>16kB</td>
<td>512B</td>
<td>2.0µA</td>
<td>$6.54</td>
</tr>
<tr>
<td>F168</td>
<td>48kB</td>
<td>2048B</td>
<td>2.0µA</td>
<td>$9.11</td>
</tr>
<tr>
<td>F1611</td>
<td>48kB</td>
<td>10240B</td>
<td>2.0µA</td>
<td>$12.86</td>
</tr>
</tbody>
</table>

TI MSP430 Microcontrollers
## Minimize Resource Use

<table>
<thead>
<tr>
<th>Model</th>
<th>ROM</th>
<th>RAM</th>
<th>Sleep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2002</td>
<td>1kB</td>
<td>128B</td>
<td>1.3µA</td>
<td>$0.94</td>
</tr>
<tr>
<td>F1232</td>
<td>8kB</td>
<td>256B</td>
<td>1.6µA</td>
<td>$2.73</td>
</tr>
<tr>
<td>F155</td>
<td>16kB</td>
<td>512B</td>
<td>2.0µA</td>
<td>$6.54</td>
</tr>
<tr>
<td>F168</td>
<td>48kB</td>
<td>2048B</td>
<td>2.0µA</td>
<td>$9.11</td>
</tr>
<tr>
<td>F1611</td>
<td>48kB</td>
<td>10240B</td>
<td>2.0µA</td>
<td>$12.86</td>
</tr>
</tbody>
</table>

**TI MSP430 Microcontrollers**

<table>
<thead>
<tr>
<th>Model</th>
<th>ROM</th>
<th>RAM</th>
<th>Sleep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM2S600</td>
<td>32kB</td>
<td>8kB</td>
<td>950µA</td>
<td>$2.73</td>
</tr>
<tr>
<td>LM3S1608</td>
<td>128kB</td>
<td>32kB</td>
<td>950µA</td>
<td>$4.59</td>
</tr>
<tr>
<td>LM3S1968</td>
<td>256kB</td>
<td>64kB</td>
<td>950µA</td>
<td>$6.27</td>
</tr>
</tbody>
</table>

**TI ARM CortexM3 Processors**
## Minimize Resource Use

<table>
<thead>
<tr>
<th>Model</th>
<th>ROM</th>
<th>RAM</th>
<th>Sleep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2002</td>
<td>1kB</td>
<td>128B</td>
<td>1.3µA</td>
<td>$0.94</td>
</tr>
<tr>
<td>F1232</td>
<td>8kB</td>
<td>256B</td>
<td>1.6µA</td>
<td>$2.73</td>
</tr>
<tr>
<td>F155</td>
<td>16kB</td>
<td>512B</td>
<td>2.0µA</td>
<td>$6.54</td>
</tr>
<tr>
<td>F168</td>
<td>48kB</td>
<td>2048B</td>
<td>2.0µA</td>
<td>$9.11</td>
</tr>
<tr>
<td>F1611</td>
<td>48kB</td>
<td>10240B</td>
<td>2.0µA</td>
<td>$12.86</td>
</tr>
</tbody>
</table>

**TI MSP430 Microcontrollers**

<table>
<thead>
<tr>
<th>Model</th>
<th>ROM</th>
<th>RAM</th>
<th>Sleep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM2S600</td>
<td>32kB</td>
<td>8kB</td>
<td>950µA</td>
<td>$2.73</td>
</tr>
<tr>
<td>LM3S1608</td>
<td>128kB</td>
<td>32kB</td>
<td>950µA</td>
<td>$4.59</td>
</tr>
<tr>
<td>LM3S1968</td>
<td>256kB</td>
<td>64kB</td>
<td>950µA</td>
<td>$6.27</td>
</tr>
</tbody>
</table>

**TI ARM CortexM3 Processors**
## Minimize Resource Use

<table>
<thead>
<tr>
<th>Model</th>
<th>ROM</th>
<th>RAM</th>
<th>Sleep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2002</td>
<td>1kB</td>
<td>128B</td>
<td>1.3µA</td>
<td>$0.94</td>
</tr>
<tr>
<td>F1232</td>
<td>8kB</td>
<td>256B</td>
<td>1.6µA</td>
<td>$2.73</td>
</tr>
<tr>
<td>F155</td>
<td>16kB</td>
<td>512B</td>
<td>2.0µA</td>
<td>$6.54</td>
</tr>
<tr>
<td>F168</td>
<td>48kB</td>
<td>2048B</td>
<td>2.0µA</td>
<td>$9.11</td>
</tr>
<tr>
<td>F1611</td>
<td>48kB</td>
<td>10240B</td>
<td>2.0µA</td>
<td>$12.86</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>ROM</th>
<th>RAM</th>
<th>Sleep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM2S600</td>
<td>32kB</td>
<td>8kB</td>
<td>950µA</td>
<td>$2.73</td>
</tr>
<tr>
<td>LM3S1608</td>
<td>128kB</td>
<td>32kB</td>
<td>950µA</td>
<td>$4.59</td>
</tr>
<tr>
<td>LM3S1968</td>
<td>256kB</td>
<td>64kB</td>
<td>950µA</td>
<td>$6.27</td>
</tr>
</tbody>
</table>

TI MSP430 Microcontrollers

TI ARM CortexM3 Processors

Sleep current necessitates microcontrollers.
Advanced applications run into ROM/RAM limits.
Two Principles

1. Minimize resource use

2. Structure code to prevent bugs
Vision
Debugging these systems is exceedingly hard.
This Talk

• Two design principles for embedded software
  ‣ Minimize resource use
  ‣ Structure interfaces and code to prevent bugs

• A technical result: static virtualization

• A lesson: avoid the island syndrome
Static Virtualization

Operating System
Static Virtualization

- Application: 3 files, 2 timers
- Operating System
Static Virtualization

Application

3 files

Operating System

2 timers
Static Virtualization

3 files

2 timers
Static Virtualization

Application

Operating System

3 files

2 timers
Static Virtualization

- Allocates exact RAM
- No pointers
- Cross-call optimization
- Dead code elimination
- Compile-time certainty
AMSenderC

Send interface

unique

unique

unique

uniqueCount
## Result

<table>
<thead>
<tr>
<th>Year</th>
<th>Version</th>
<th>Multihop yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003&lt;sup&gt;a&lt;/sup&gt;</td>
<td>TinyOS 0.6</td>
<td>58%</td>
</tr>
<tr>
<td>2005&lt;sup&gt;b&lt;/sup&gt;</td>
<td>TinyOS 1.1</td>
<td>68.5%</td>
</tr>
<tr>
<td>2009&lt;sup&gt;c&lt;/sup&gt;</td>
<td>TinyOS 2.0</td>
<td>99.58%</td>
</tr>
</tbody>
</table>

  “The multi-hop burrow motes perform worse (with a median yield of 58%) but within tolerance”

  “the median event yield was 68.5%” *(events, not packets)*

  “the system achieved a median network reliability of 99.68% (range 95.2% – 100%). In contrast, the sensing reliability was significantly lower.”
## Result

<table>
<thead>
<tr>
<th>Year</th>
<th>Version</th>
<th>Multihop yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>TinyOS 0.6</td>
<td>58%</td>
</tr>
<tr>
<td>2005</td>
<td>TinyOS 1.1</td>
<td>68.5%</td>
</tr>
<tr>
<td>2009</td>
<td>TinyOS 2.0</td>
<td>99.58%</td>
</tr>
</tbody>
</table>

Static virtualization enabled applications to be highly robust, dependable, and efficient.

---


*b Werner-Allen et al. “Fidelity and Yield in a Volcano Monitoring Sensor Network.” OSDI 2006. “the median event yield was 68.5% (events, not packets)”

*c Chipara et al. “Reliable Clinical Monitoring using Wireless Sensor Networks: Experiences in a Step-down Hospital Unit.” Sensys 2010. “the system achieved a median network reliability of 99.68% (range 95.2% – 100%). In contrast, the sensing reliability was significantly lower.”
This Talk

• Two design principles for embedded software
  ‣ Minimize resource use
  ‣ Structure interfaces and code to prevent bugs

• A technical result: static virtualization

• A lesson: avoid the island syndrome
Applications
Research vs. Practice

- TinyOS technically focused on enabling users to build larger, more complex applications
- Doing so increased the learning curve to building simple ones
Timeline

- First TinyOS code
- ASPLOS paper
- NEST bootcamp
- v0.6
- v1.0
- v1.1
- Core WG forms
- v1.15
- NEST concludes
- v2.0 beta1
- v2.0 beta2
- v2.0
- v2.0.1
- v2.0.2
- v2.1
- v2.1.1

- nesC, bidirectional and parameterized interfaces
- atomic, uniqueCount
- generics, nx_types
- safe, threads
Statically Virtualized Timer
(TinyOS 2.x)

AppP.nc

```java
Control.start() {
    Timer.start(..)
}

Timer.fired() {
    send_packet();
}
```

AppC.nc

```java
T = new TimerC()
AppP.Timer -> TimerC.Timer
```
Implementation
(TinyOS 2.x)

AppP.nc

Control.start() {
  Timer.start(..)
}

Timer.fired() {
  send_packet();
}

AppC.nc

T = new TimerC()
AppP.Timer -> TimerC.Timer

clock_interrupt {
  update_ts()
  for i = 0 to NT-1
    if (ts[i].fire)
      Timer[i].fired();
}

Timer[i].start(...) {
  startTimer(i, ..)
}

TimerC.nc

#define TS unique("T")
TimerC.Timer = TimerP.Timer[TS]

TimerP.nc

#define NT uCount("T")
timer_t ts[NT];
clock_interrupt {
  update_ts()
  for i = 0 to NT-1
    if (ts[i].fire)
      Timer[i].fired();
}

Timer[i].start(...) {
  startTimer(i, ..)
}
#define TS unique("T")
TimerC.Timer = TimerP.Timer[TS]

#define NT uCount("T")
timer_t ts[NT];
clock_interrupt {
  update_ts()
  for i = 0 to NT-1
    if (ts[i].fire)
      Timer[i].fired();
}

Timer[i].start(...) {
  startTimer(i, ..)
}

Control.start() {
  Timer.start(..)
}
Timer.fired() {
  send_packet();
}
APP.c

APP_START() {
    APP_TIMER_INIT(...);
}

APP_TIMER() {
    send_packet();
}

APP.desc

APP_TIMER_INIT  TIMER_START
TIMER_FIRE  APP_TIMER

TIMER.c

timer_t ts;

TIMER_START(...) {
    set_ts();
    init_interrupt();
}

clock_interrupt {
    update_ts()
    TIMER_FIRE();
}
Code Evolution

• Code evolved to use nesC features in more complex and intricate ways
  ‣ Improved software dependability
  ‣ Allowed more complex applications
  ‣ Served the needs of the community

• Increased barrier to entry: island syndrome
Death by Components

- Fine-grained component toolkits are great for building and evolving a system
- The end result is difficult for a new user to understand: increases the learning curve
- Need to transition to structurally simpler implementations
Death by Components

- TimerC
- HplTimer0C
- AlarmCounterP
- AlarmSyncC
- AlarmAsyncP
- HplTimer0P
- McuSleepC
- CounterToTimeC
- AlarmToTimerC
- VirtualizeTimerC
Another Approach
Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It’s intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software running on a computer (e.g. Flash, Processing, MaxMSP).
• **Tremendously successful academic project**
  ‣ Started as Perl scripts used by a handful of academics
  ‣ Now ~100 downloads a day, hundreds of thousands of nodes
  ‣ Has a worldwide community of hundreds of contributors

• **But it could have been more so**
  ‣ Missed being a platform for simple sensing apps (Arduino)
  ‣ Missed being a platform for the Internet of Things (Contiki)
  ‣ “Applications” became “hard applications”
  ‣ Should have focused on the simple as much as the complex (the island syndrome)
Disclaimer

TinyOS is the work of hundreds of contributors over a decade.
(of which I am only one, the core WG chair, who joined 18 months in)

This paper and talk are my personal opinions and observations.
TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters. A worldwide community from academia and industry use, develop, and support the operating system as well as its associated tools, averaging 35,000 downloads a year.

**Latest News**

**August 20, 2012:** TinyOS 2.1.2 is now officially released; you can download it from the debian packages on tinyos.stanford.edu. Manual installation with RPMs with the instructions on docs.tinyos.net will be forthcoming. TinyOS 2.1.2 includes:

- Support for updated msp430-gcc (4.6.3) and avr-gcc (4.1.2).
- A complete 6lowpan/RPL IPv6 stack.
- Support for the ucmni platform and ATmega128RFA1 chip.
- Numerous bug fixes and improvements.

**July 21, 2010:** The transition from hosting TinyOS at Sourceforge to Google code is now complete. Part of this transition included placing all of TinyOS under a New BSD license (in Sourceforge several compatible licenses were used). Thanks to all of the developers for agreeing to move to a uniform license!

**FAQ**
- Frequently asked questions about TinyOS

**Learn**
- Download TinyOS and learn how to use it

**Community**
- TinyOS Working Groups, mailing lists, and TEPs

TinyOS is also deeply indebted to its users, their bug reports, feature requests, and hard work.